Yasmin Samy

Ch 7
lec (-)

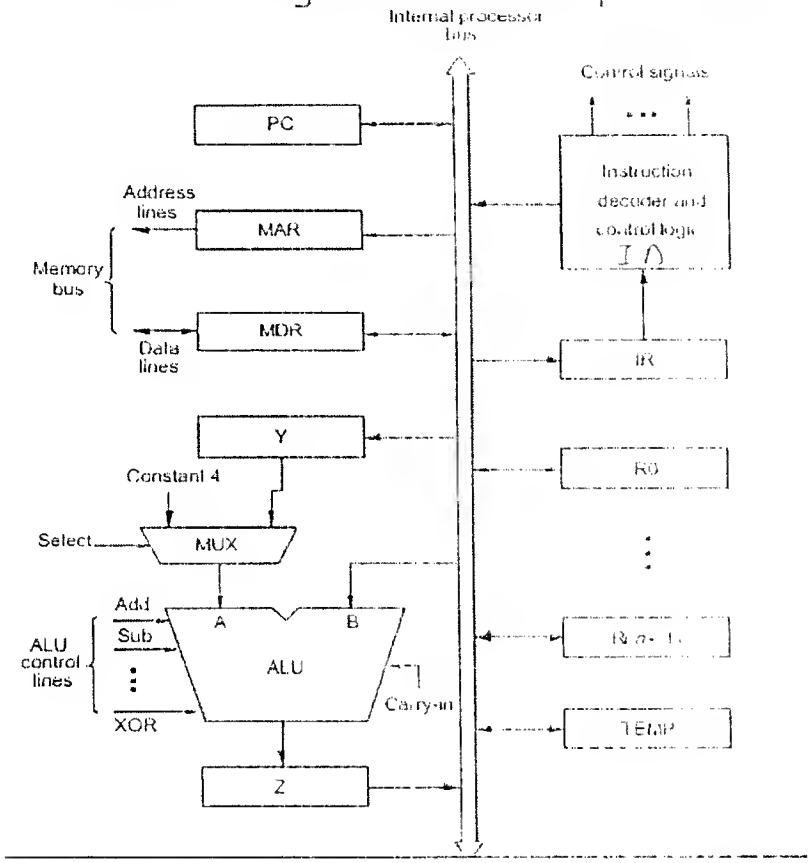
Chapter 3 (4)

CPU Architecture

lec 3 (A, B)
Ch 7

Q1: discuss how Single bus organization connects the computer system. List its disadvantages.

Internal organization of a processor



a processor has several registers/building blocks:

- Memory address register (MAR)
- Memory data register (MDR)
- Program Counter (PC)
- Instruction Register (IR)
- General purpose registers R0 - R(n-1)
- Arithmetic and logic unit (ALU)
- Control unit.

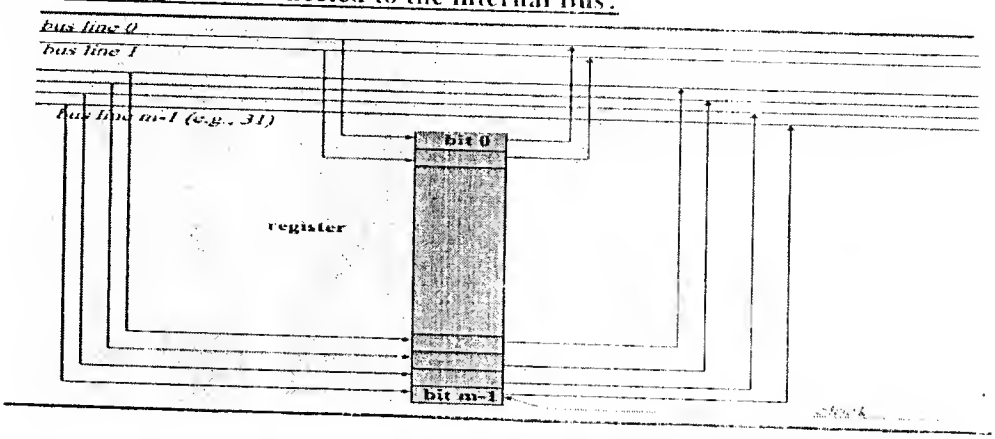
(Recall from Chapter 2):

- **Single bus organization:**
  - ALU, control unit and all the registers are connected via a single common bus.
  - Bus is internal to the processor and should not be confused with the external bus that connects the processor to the memory and I/O devices.
- **Data lines** of the external memory bus are connected to the internal processor bus via MDR
  - ✓ Register MDR has two inputs and two outputs.
  - ✓ Data may be loaded to (from) MDR from (to) internal processor bus or external memory bus.
- **Address lines** of the external memory bus are connected to the internal processor bus via MAR.
  - ✓ MAR receives input from the internal processor bus.
  - ✓ MAR provides output to external memory bus.
- **Instruction decoder and control logic block, or control unit** issues signals to control the operation of all units inside the processor and for interacting with the memory bus.
  - ✓ Control signals depend on the instruction loaded in the Instruction Register (IR)
- **Outputs from the control logic block are connected to:**
  - ✓ Control lines of the memory bus.
  - ✓ ALU, to determine which operation is to be performed.
    - Select input of the multiplexer MUX to select between Register Y and constant 4.
  - ✓ Control lines of the registers, to select the registers.
- **Registers Y, Z, and TEMP:**
  - ✓ Used by the processor for temporary storage during execution of some instructions.
  - ✓ Note that Registers R0 to R(n-1) are used to store data generated by one instruction for later use by another instruction.
  - ✓ Data is stored in R0 through R(n-1) after the execution of an instruction.
- **Multiplexer MUX** selects either the output of register Y or a constant 4, depending upon the control input Select.
  - ◆ Constant 4 is used to increment the value of the PC.

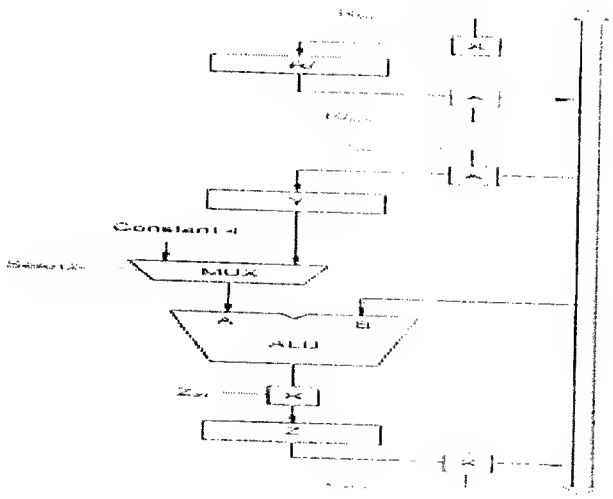
⇒ **Disadvantages of using Simple single-bus structure:**

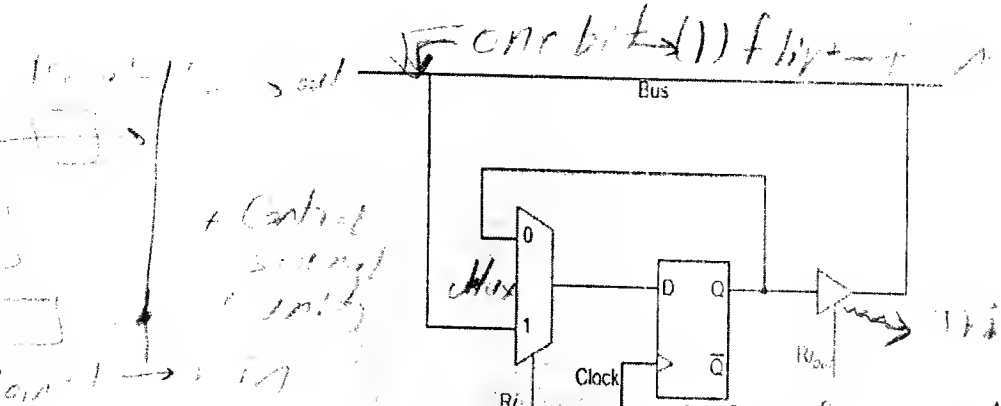
Results in long control sequences, because only one data item can be transferred over the bus in a clock cycle

Q2: How are Registers connected to the internal Bus?



- ✓ At any one time (rising edge of clock pulse), only one register may output its contents to the bus.
- ✓ Registers load data by control signal.
- ✓ Registers are connected to the bus via switches controlled by the signals **Rin** & **Rout**.
- ✓ Each register **Ri** has two control signals: **Ri in and Ri out.**
- ✓ If Ri in=1, the data from the bus is loaded into the register.
- ✓ If Ri out=1, the data from the register is loaded onto the bus.
- ✓ The same holds for registers Y and Z as well.



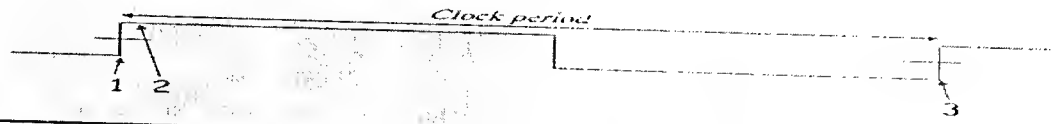


- ✓ **Each bit in a register** may be implemented **by an edge-triggered D-flip flop**.
- ✓ **Two input multiplexer** is used to select the data applied to the input of an edge triggered flip-flop.
- ✓ **Q output of the flip-flop** is connected to the bus via a tri-state gate.

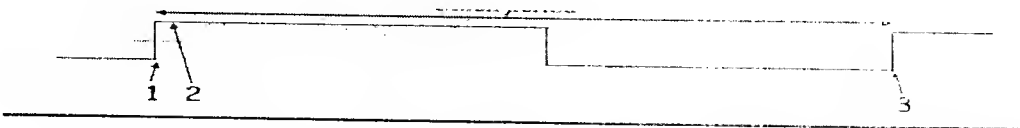
Ri in	Ri out
✓ **If Ri in=1:** ✓ Multiplexer selects the data on the bus. ✓ Data is loaded into the flip-flop at the rising edge of the clock. ✓ **if Ri in=0:** ✓ Multiplexer feeds back the value currently stored in the flip-flop. ✓ Q output represents the value currently stored in the flip-flop.	✓ **If Ri out=1:** ▪ Tri-state gate loads the value of the flip-flop onto the bus. ▪ Data (Q) is loaded onto the bus at the rising edge of the clock. ✓ **If Ri out=0:** ▪ Gate's output is in **high impedance** (electrically disconnected) state. ▪ Corresponds to open-circuit state.

- ✓ **Data transfers and operations** take place within time periods defined by the processor clock.
- ✓ **multiphase clocking:** When edge-triggered flip-flops are not used, two or more clock signals may be needed to guarantee proper transfer of data.

Q3: How register transfer works with clock pulse. Assume you need to transfer data from register Transfer the contents of register R3 to register R4.



1. Control signals $R3_{out}$ and $R4_{in}$ become 1. They stay valid until the end of the clock
2. After a small delay, the contents of R3 are placed onto the bus. The contents of R3 stay onto the bus until the end of the clock cycle.
3. At the end of the clock cycle, the data onto the bus is loaded into R4. $R3_{out}$ and $R4_{in}$ become 0.

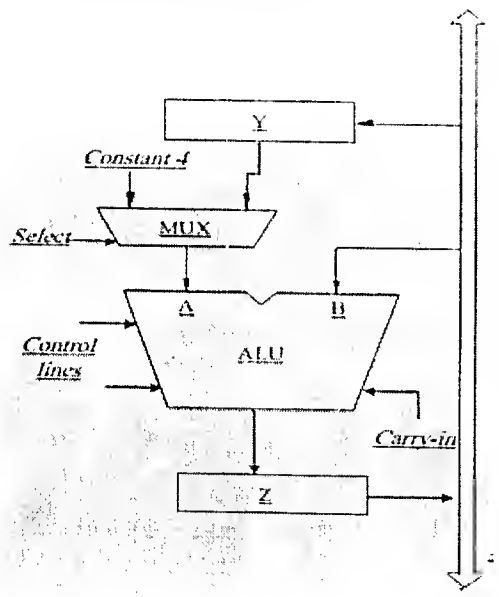


1. Control signals R3out, R4in and R5in become1 They stay valid until the end of the clock cycle.
2. After a small delay, the contents of R3are placed onto the bus. The contents of R3 stay onto the bus until the end of the clock cycle.
3. At the end of the clock cycle, the data onto the bus is loaded into R4 and R5.R3 out, R4in and R5in become 0.

Notes

- **The number of registers that can be simultaneously loaded depends on:**
  - Drive capability (fan-out)
  - Noise.
  - Note that this is an electrical issue, not a logical issue.

Q4: Performing an arithmetic operation Add the contents of registers *R1* and *R2* and place the result in *R3*. That is: *R3 = R1 + R2*.



1. Place the contents of register R1 into the Y register in the first clock cycle.

2. Place the contents of register R2 onto the bus in the second clock cycle. Both inputs to the ALU are now valid. Select register Y, and assert the ALU command F=A+B.

3. In the third clock cycle, Z register has latched the output of the ALU. Thus the contents of the Z register can be copied into register R3.

Clock Cycle 1:

$R1_{out}, Y_{in}$ (Y=R1)

Clock Cycle 2:

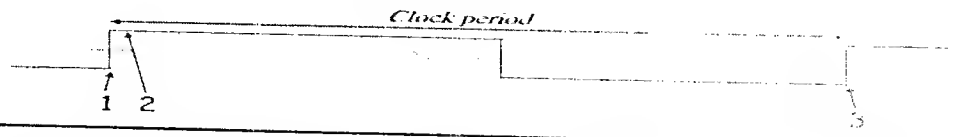
$R2_{out}$, Select Y, Add, Z_{in} (Z = R1+R2)

Clock Cycle 3:

$Z_{out}, R3_{in}$ (R3=Z)

Q4:How register transfer works with clock pulse. Assume you need to transfer data from register

Transfer the contents of register R3 to register R4, R5.

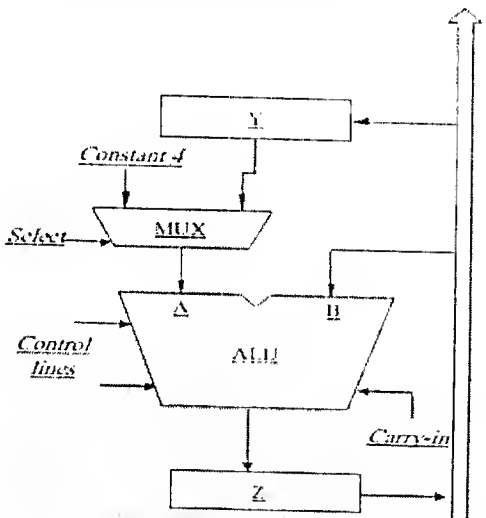


1. Control signals R3out, R4in and R5in become1. They stay valid until the end of the clock cycle.
2. After a small delay, the contents of R3are placed onto the bus. The contents of R3 stay onto the bus until the end of the clock cycle.
3. At the end of the clock cycle, the data onto the bus is loaded into R4 and R5.R3 out, R4in and R5in become 0.

Notes

- **The number of registers that can be simultaneously loaded depends on:**
  - Drive capability (fan-out)
  - Noise.
  - Note that this is an electrical issue, not a logical issue.

Q4: Performing an arithmetic operation Add the contents of registers *R1* and *R2* and place the result in *R3*. That is: *R3 = R1 + R2.*



1. Place the contents of register R1 into the Y register in the first clock cycle.

2. Place the contents of register R2 onto the bus in the second clock cycle. Both inputs to the ALU are now valid. Select register Y, and assert the ALU command F=A+B.

3. In the third clock cycle, Z register has latched the output of the ALU. Thus the contents of the Z register can be copied into register R3.

Clock Cycle 1:

$R1_{out}, Y_{in}$ (Y=R1)

Clock Cycle 2:

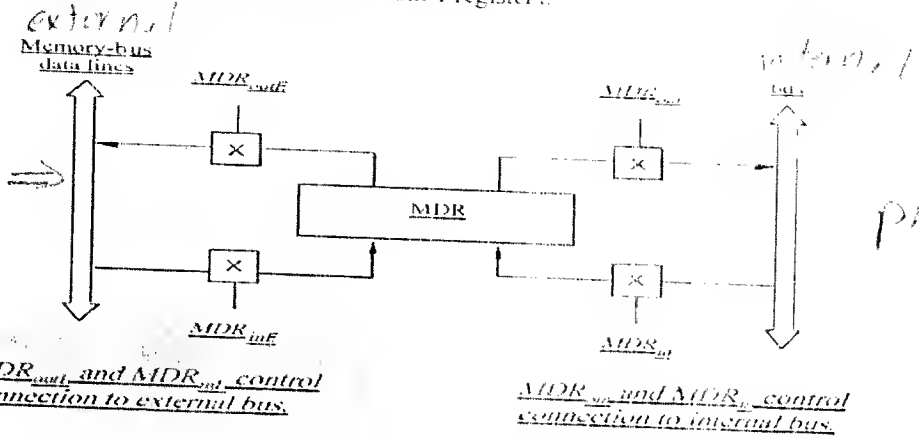
$R2_{out}$, Select Y, Add, Z_{in} (Z = R1+R2)

Clock Cycle 3:

$Z_{out}, R3_{in}$ (R3=Z)

Q5: How can word fetched from memory? What is the required signal that the processor waits from memory to know that the word is fetched? Give Example.

- Processor has to specify the address of the memory location where this information is stored and request a Read operation.
- Processor transfers the required address to *MAR*.
  - Output of *MAR* is connected to the address lines of the memory bus.
- Processor uses the control lines of the memory bus to indicate that a *Read* operation is needed.
- Requested information is received from the memory and is stored in *MDR*.
  - Transferred from *MDR* to other registers.



- Timing of the internal processor operations must be coordinated with the response time of memory Read operations.
- Processor completes one internal data transfer in one clock cycle.
- Memory response time for a Read operation is variable and usually longer than one clock cycle.
  - Processor waits until it receives an indication that the requested Read has been completed.
  - **Control signal Memory Function Completed (MFC)** is used for this purpose.
  - **MFC is set to 1** by the memory to indicate that the contents of the specified location have been read and are **available** on the data lines of the memory **bus**.

Example 1: ***List the steps to execute this read from memory MOVE (R1), R2***

1. Load the contents of Register *R1* into *MAR*.
2. Start a *Read* operation on the memory bus.
3. Wait for *MFC* response from the memory
4. Load *MDR* from the memory bus.
5. Load the contents of *MDR* into Register *R2*.

1. **Steps 1 and 2 can be combined.**

- Load *R1* to *MAR* and activate *Read* control signal simultaneously.
- $R1_{out}$, MAR_{in}, *Read*.

2. **Steps 3 and 4 can be combined.**

- Activate control signal MDR_{inE} while waiting for response from the memory MFC.
- MDR_{inE}, *WMFC*

3. **Last step**

- Loads the contents of *MDR* into Register *R2*. Hence, Memory *Read* operation takes 3 steps.
- MDR_{outB}, $R2_{in}$

ملخص ما سبق

1. $R1_{out}$, MAR_{in}, *Read*.
2. MDR_{inE}, *WMFC*
3. MDR_{outB}, $R2_{in}$

Q6 : Write the steps of executing this instruction of writing word to memory:

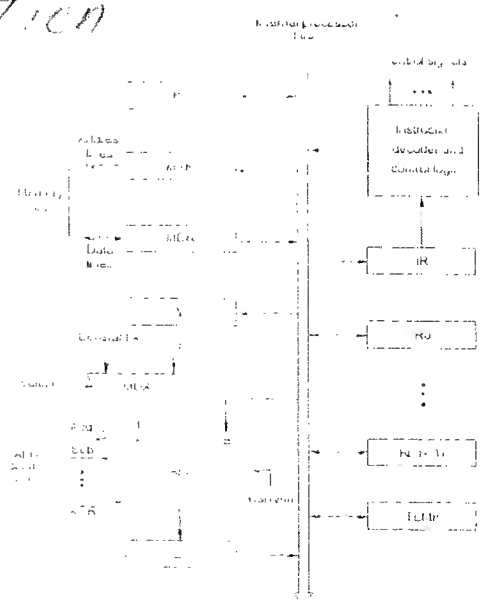
MOVE R2, (R1):

1. $R1_{out}$, MAR_{in}
2. $R2_{out}$, MDR_{in}, *Write*
3. MDR_{outE}, *WMFC*

Q7: Example 3: Execution of a complete instruction(Fetch→Decode→Execute)

ADD (R3), R1

Step	Action
1	PC_{out}, MAR_{in}, *Read*, *Select4*, *Add*, Z_{in}
2	Z_{out}, PC_{in}, *WMFC*
3	MDR_{out}, IR_{in}
4	$R3_{out}$, MAR_{in}, *Read*
5	$R1_{out}$, Y_{in}, *WMFC*
6	MDR_{out}, *SelectY*, *Add*, Z_{in}
7	Z_{out}, $R1_{in}$, *End*



Q8: List steps of Unconditional Branch instructions

- Branch target address is computed by adding the updated contents of the *PC* to an offset.
- Copying the updated contents of the *PC* to Register *Y* speeds up the execution of *BRANCH* instruction.
- Since the Fetch cycle is the same for all instructions, this step is performed for all instructions.

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Offset-field-of-IR_{out}, Add, Z_{in}
5	Z_{out}, PC_{in}, End

Figure 7.7 Control sequence for an unconditional Branch instruction.

Example: Conditional Branch instructions

Consider now a conditional branch. In this case, we need to check the status of the condition codes before loading a new value into the PC. For example, for a Branch-on-negative (Branch<0) instruction, step 4 in Figure 7.7 is replaced with

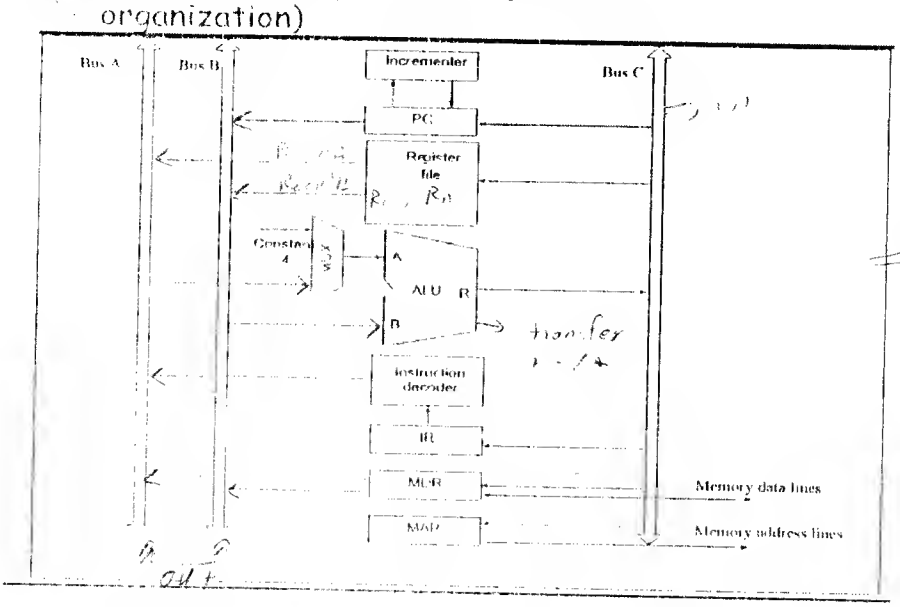
4	Offset-field-of-IR_{out}, Add, Z_{in},	If N = 0 then End

Thus, if N=0 the processor returns to step 1 immediately after step 4. If N=1, step 5 is performed to load a new value into the PC, thus performing the branch operation.

Q8: why we need the multiple bus organization. Draw its architecture.

- Most commercial processors provide multiple internal paths to enable several transfers to take place in parallel.

Multiple bus organization (Three-bus organization)



- Three-bus organization to connect the registers and the ALU of a processor.
- All general purpose registers are combined into a single block called register file.
  - Register file has three ports.
  - *Two outputs ports connected to buses A and B*, allowing the contents of two different registers to be accessed simultaneously, and placed on buses A and B.
  - *Third input port allows the data on bus C* to be loaded into a third register during the same clock cycle.
- **Inputs to the ALU and outputs from the ALU:**
  - Buses A and B are used to transfer the source operands to the A and B inputs of the ALU.
  - Result is transferred to the destination over bus C.
- **ALU can also pass one of its two input operands unmodified if needed:**
  - Control signals for such an operation are *R=A* or *R=B*.

- **Incrementer unit:**
  - Used to increment the PC by 4.
  - Source for the constant 4 at the ALU multiplexer can be used to increment other addresses such as the memory addresses in multiple load/store.

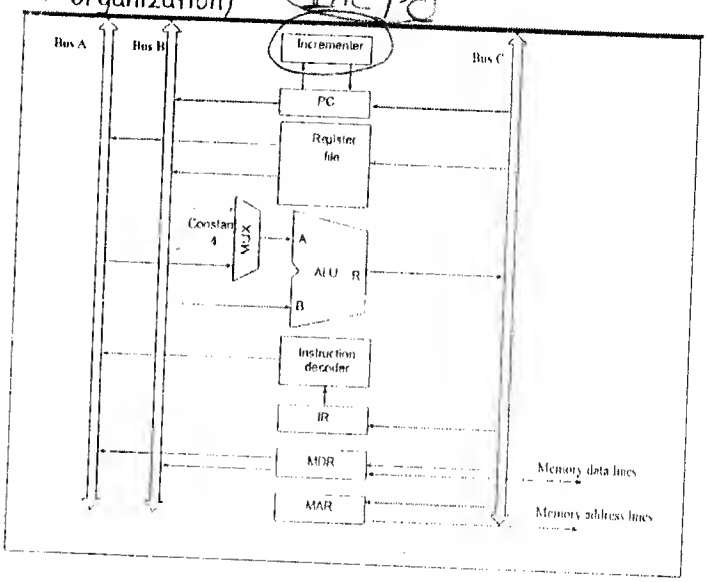
Example: Three operand instruction: *ADD R4, R5, R6*

Three operand instruction: *ADD R4, R5, R6*

Step	Action
1	PC_{out}, R=B, MAR$_{in}$, Read, IncPC
2	WMFC
3	MDR_{outB}, R=B, IR$_{in}$
4	$R4_{outA}$, $R5_{outB}$, SelectA, Add, $R6_{in}$, End

1. Pass the contents of the PC through ALU and load it into MAR. Increment PC.
2. Wait for MFC.
3. Load the data received into MDR and transfer to IR through ALU.
4. Execution of the instruction is the last step

Multiple bus organization (Three-bus organization)



Q2: How the Control Unit generates the proper control signal for each instruction?

To execute instructions the processor must generate the necessary control signals in proper sequence by.

1. **Hardwired control:**
   - ✓ Control unit is designed as a finite state machine.
   - ✓ Inflexible but fast.
   - ✓ Appropriate for simpler machines (e.g. RISC machines)
2. **Microprogrammed control:**
   - ✓ Control path is designed hierarchically using principles identical to the CPU design.
   - ✓ Flexible, but slow
   - ✓ Appropriate for complex machines (e.g. CISC machines)

Q10: List all the details about how the hardwired control generates the required Control Signals. Give Example.

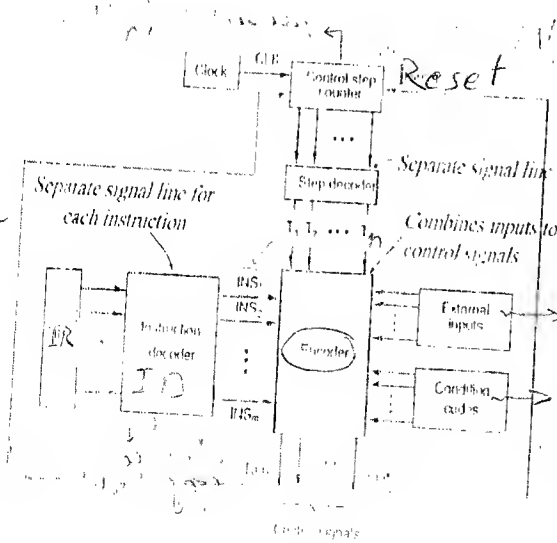
Step	Action
1	PC_{out}, MAR_{in}, Read, Select4Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R3_{out}$, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End

- ✓ Each step in this sequence is completed in one clock cycle.
- ✓ A counter may be used to keep track of the control steps.
- ✓ Each state or count, of this counter corresponds to one control step.

Required control signals are determined by the following information:

1. **Contents of the control step counter.**
   - Determines which step in the sequence.
2. **Contents of the instruction register.**
   - Determines the actual instruction
3. **Contents of the condition code flags.**
   - Used for example in a BRANCH instruction.
4. **External input signals such as MFC.**

Control unit organization



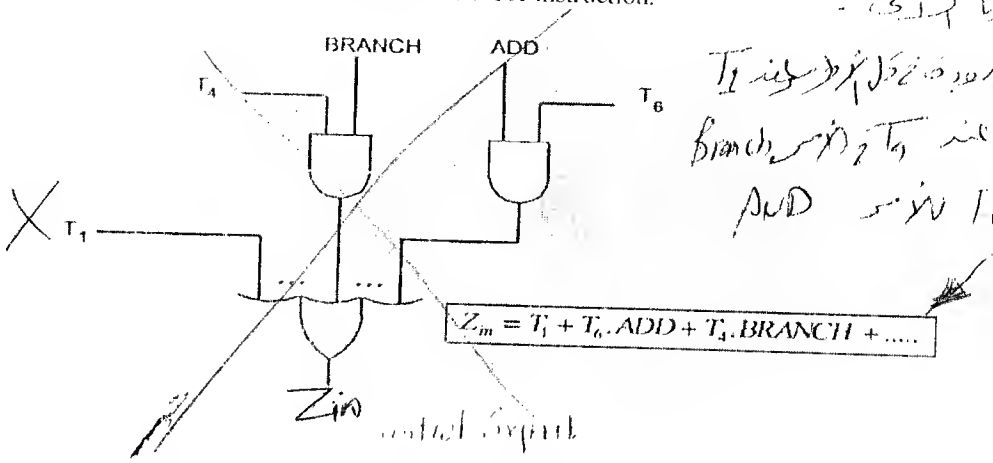
Control unit consists of a decoder/encoder block to accept the following inputs:

- ✓ Control step counter.
- ✓ Instruction Register(IR)
- ✓ Condition codes.
- ✓ External inputs.

CU Generates control *signals*.

Example: How Control signals such as Z_{in}, PC_{out}, ADD are generated by encoder block in the hardwired control circuit.

- **Suppose if Z_{in} is asserted:**
  - ✓ During T_1 for all instructions.
  - ✓ During T_6 for *ADD* instruction.
  - ✓ During T_4 for unconditional *BRANCH* instruction.



- **Control hardware can be viewed as a state machine:**
  - Changes state every clock cycle depending on the contents of the instruction register, condition codes, and external inputs.
- **Outputs of the state machine are control signals:**
- Sequence of control signals generated by the machine is determined by wiring of logic elements, hence the name "hardwired control".

An advantage hardwired control of Speed of operation is one of the advantages of hardwired control is its speed of operation.

Disadvantages of hardwired control include:

- Little flexibility.
- Limited complexity of the instruction set it can implement.

Q11: Define Control Word, Microroutine, Control Store, and Microprogram Counter (Mpc).

Control Word (CW)= microinstructions

- **CW** is a word whose individual bits represent various control signals.
- Every instruction will have its own microroutine which is made up of microinstructions or CW.
- *At every step, a Control Word needs to be generated.*
- *Each CW in this microroutine is referred to as a* ***microinstruction.***

Microroutine:

- ✓ *Every **instruction** will need a sequence of CWs for its execution.*
- ✓ *Sequence of CWs for an **instruction** is the **microroutine** for the instruction.*

Control Store: the microroutines for all instructions in the instruction set of a computer are stored in a special memory called Control Store.

Microprogram Counter (Mpc): is something different than the program counter (PC) of the microprocessor. Mpc is responsible about the generation of T1, T2,T3,.....Tn control steps may take different clock cycles

Example:

Microprogrammed control (contd..)

Control Word (CW) is a word whose individual bits represent various control signals

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMF
3	MDR_{out}, IR_{in}
4	$R3_{out}$, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End

At every step, some control signals are asserted (=1) and all others are 0

Control Signals:
PC_{out}
PC_{in}
MAR_{in}
Read
MDR_{out}
IR_{in}
Y_{in}
SelectY
Select4
Add
Z_{in}
Z_{out}
$R1_{out}$
$R1_{in}$
$R3_{out}$
WMFC
End

Microroutine

Microprogrammed control (contd..)

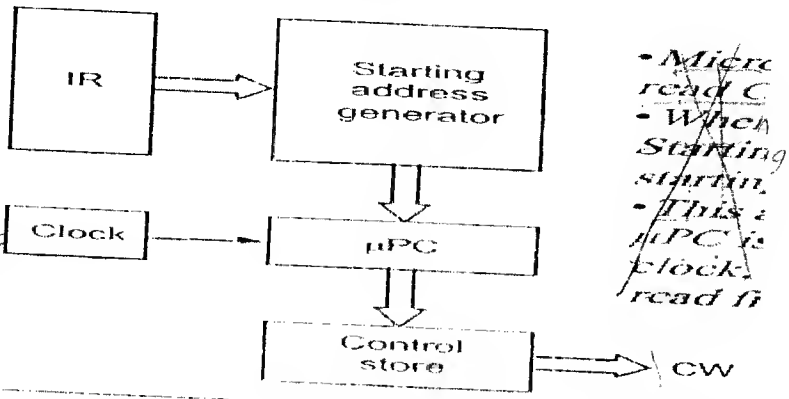
Control Word

Micro-instruction	PC$_{in}$	PC$_{out}$	MAR$_{in}$	Read	MDR$_{out}$	IR$_{in}$	Y$_{in}$	Select	Add	Z$_{in}$	Z$_{out}$	R1$_{out}$	R1$_{in}$	R3$_{out}$	WMFC	End
1	0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0
2	1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0
3	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
4	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0
5	0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0
6	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1

- *At every step, a Control Word needs to be generated.*
- *Every **instruction** will need a sequence of CWs for its execution.*
- *Sequence of CWs for an instruction is the **microroutine** for the instruction.*
- *Each CW in this microroutine is referred to as a **microinstruction**.*

(SelectY is represented by Select=0, & Select4 by Select=1)

Q12:Draw the Basic organization of a microprogrammed control unit. Then explain How the microprogramming control generates control signal



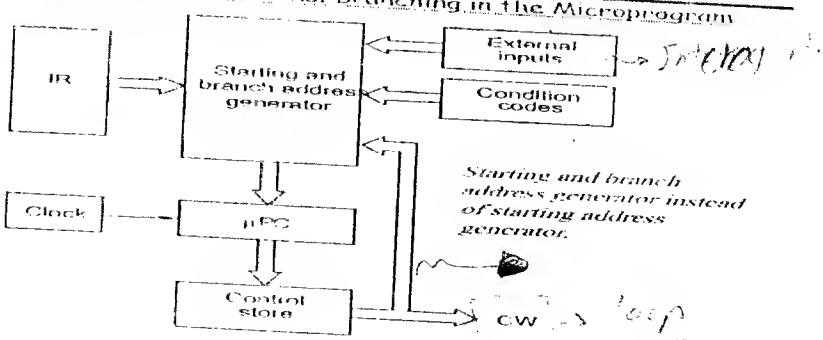
1. *Microprogram counter (mPC)* is used to read CWs from control store sequentially.
2. When a new instruction is loaded into IR, **Starting address generator** generates the starting address of the microroutine.
3. This address is loaded into the mPC. mPC is automatically incremented by the clock, so successive microinstructions are read from the control store.

Q13: Basic organization of the microprogrammed control unit cannot check the status of condition codes or external inputs to determine what should be the next microinstruction. How to handle this in microprogrammed control. Give Example then Explain how this works.

- **To handle this in microprogrammed control:**
  - ✓ Use conditional branch microinstructions.
  - ✓ These microinstructions, in addition to the branch address also specify which of the external inputs, condition codes or possibly registers should be checked as a condition for branching.

Microprogrammed control (contd..)

Control Unit with Conditional Branching in the Microprogram



- ✓ **Starting and branch address generator** accepts as inputs:
  - Contents of the Instruction Register **IR** (as before).
  - External inputs
  - Condition codes
  - ✓ Generates a new address and loads it into microprogram counter (*mPC*) when a microinstruction instructs it do so.
  - ✓ *mPC* is incremented every time a microinstruction is fetched except:
    - New instruction is loaded into IR, *mPC* is loaded with the starting address of the microroutine for that instruction.
    - Branch instruction is encountered and branch condition is satisfied, *mPC* is loaded with the branch address.
    - End instruction is encountered, *mPC* is loaded with the address of the first CW in the microroutine for the instruction fetch cycle.

Example: Branch handling--- microprogram control

Microroutine for the instruction (Branch < 0)

Address	Microinstruction	
0	PC_{out}, MAR_{in}, Read, SelectM, Add, Z_{in}	Fetch *BRANCH<0* instruction, microroutine is at address 25.
1	Z_{out}, PC_{in}, Y_{in}, WMFC	
2	MDR_{out}, IR_{in}	
3	Branch to starting address of appropriate microroutine	Branch to address 25.
25	If N=0, then branch to microinstruction 0	Test the N bit of the condition codes If 0, go to 0 and get new instr. Else execute microinstruction located at 26 and put the branch target address into Register Z. (Microinstruction at location 27).
26	Offset-field-of-IR_{out}, SelectY, Add, Z_{in}	
27	Z_{out}, PC_{in}, End	

Address 25 is the output of starting address generator and is loaded into the microprogram counter (*μPC*).

Q14: How the Microinstruction is formatted.

1-Simple approach is to allocate one bit for each control signal Results in long microinstructions.

✓ **Disadvantages:**

✓ Since the number of control signals is usually very large few bits are set to 1 in any microinstruction, resulting in a poor use of bit space.

✓ **Solution**

✓ Reduce the length of the microinstruction by taking advantage of the fact that most signals are not needed simultaneously, and many signals are mutually exclusive.

For example of disadvantages:

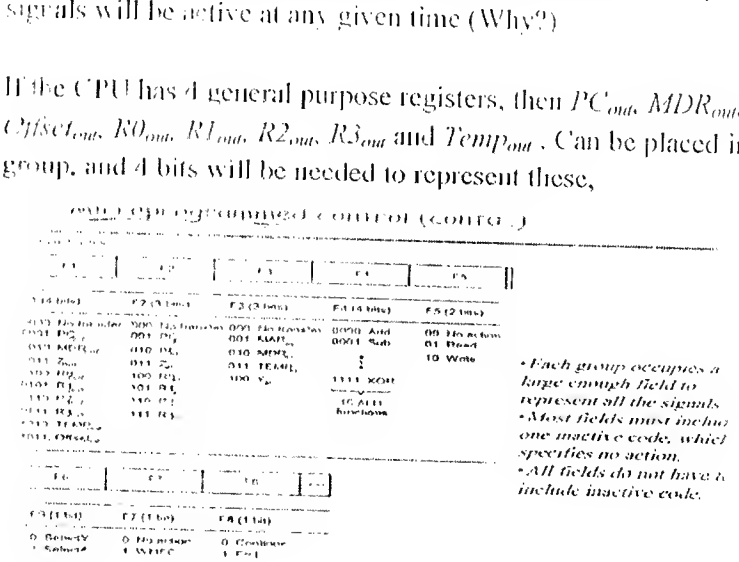
a. Only one ALU function is active at a time.
b. Source for a data transfer must be unique.
c. *Read* and *Write* memory signals cannot be active simultaneously.

2-**Group mutually exclusive signals** in the same group.

✓ At most one micro operation can be specified per group.
✓ Use binary coding scheme to represent signals within a group.

Example of Grouping control signals:

✓ If ALU has 16 operations, then 4 bits can be sufficient.
✓ Group register output signals into the same group, since only one of these signals will be active at any given time (Why?)
✓
✓ If the CPU has 4 general purpose registers, then PC_{out}, MDR_{out}, Z_{out}, $Offset_{out}$, $R0_{out}$, $R1_{out}$, $R2_{out}$, $R3_{out}$ and $Temp_{out}$. Can be placed in a single group, and 4 bits will be needed to represent these,



Q15: Define the Vertical and Horizontal Organization of micro-program.

In vertical organization :

✓ Each microinstruction contains a small number of control functions leading to more microinstructions required for the execution of each instruction.

- ✓ Results in slower operating speeds.

In horizontal organization :

- ✓ Many resources can be controlled with a single instruction leading into small number of microinstructions required for the execution of each instruction.
- ✓ Require that the CPU structure allows parallel use of resources.
- ✓ results in higher operating speed